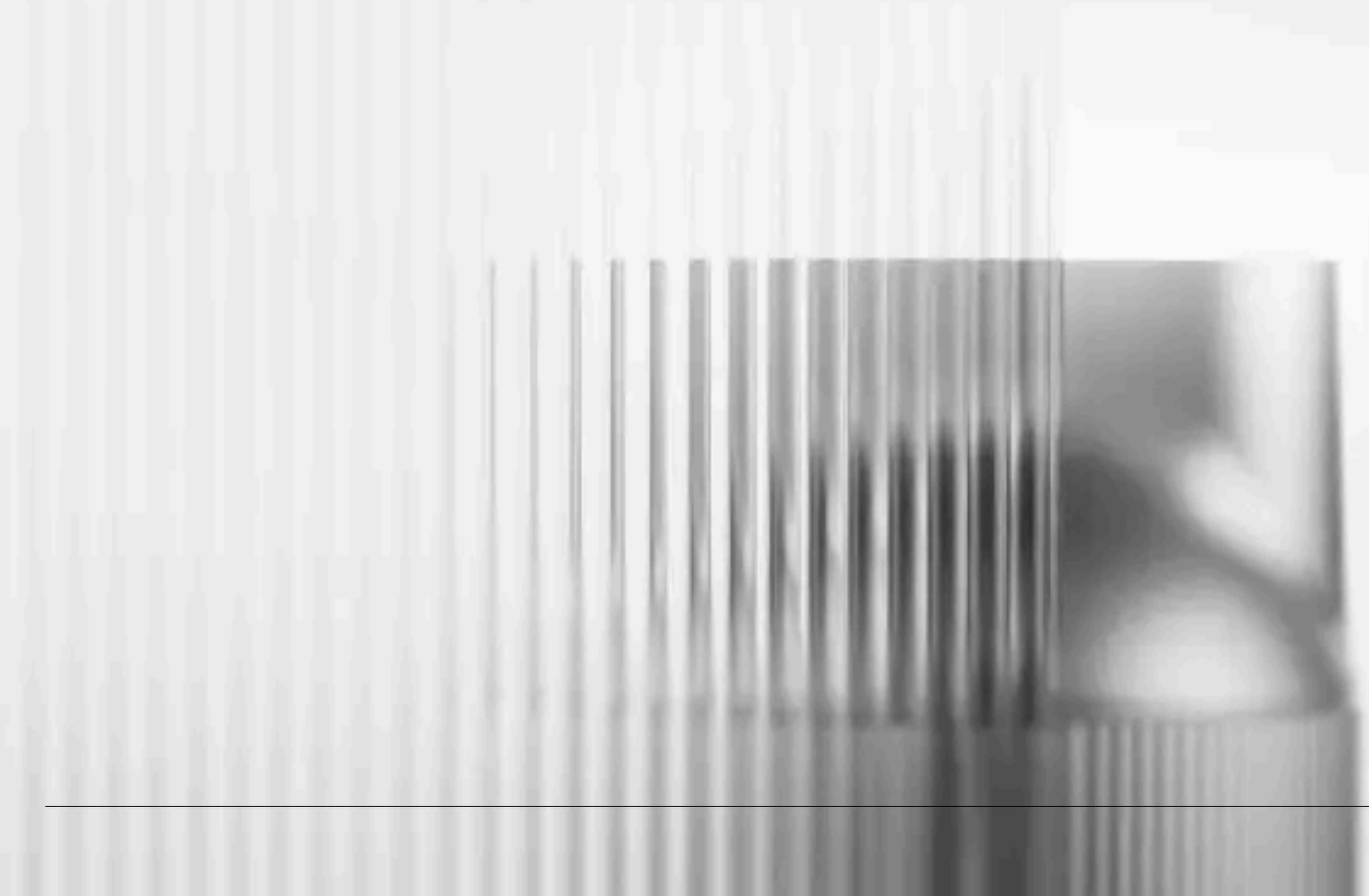
INDITEXTECH

2026

SCALING CEPHFS MDS
FOR FRONT DEVELOPER WORKSPACE ON KUBERNETES



INDITEXTECH

AGENDA

01 INDITEX

02 ARQUITECTURE

03 BOTTLENECK

04 SCALE UP

05 OBSERVABILITY

06 OPEN CASES & LESSONS

INDITEX

01


ZARA
1975


PULL&BEAR
1991


Massimo Dutti
1991


BERSHKA
1998

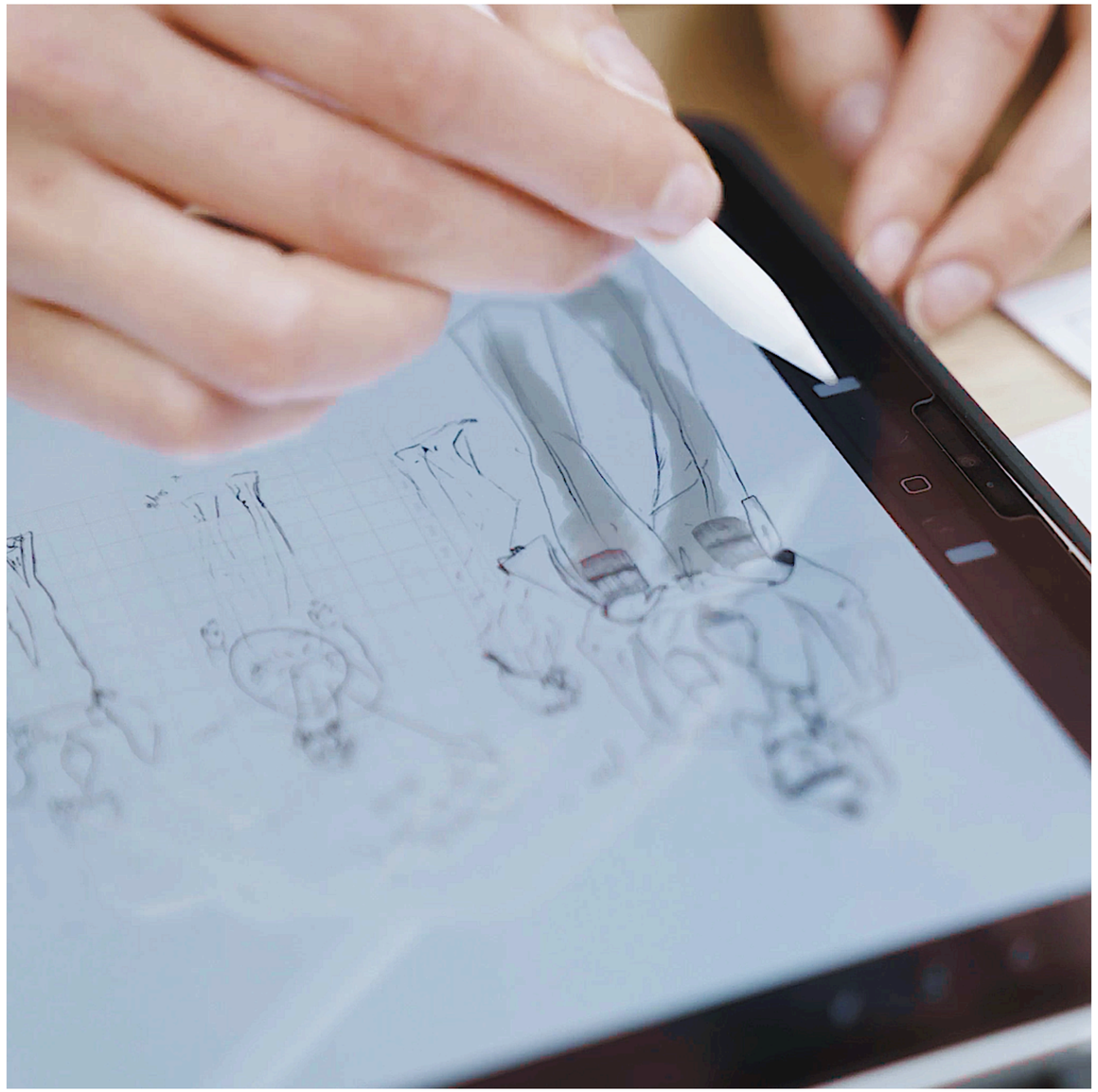

STRADIVARIUS
1999


lefties
1999


OYSHO
2001


ZARA HOME
2003

WE ASPIRE TO EVOKE EMOTIONS THROUGH IMAGERY, DESIGN, AND TECHNOLOGY



DATA

INDITEX

8
BRANDS

+200
MARKETS

+5.400
STORES

TECH AREA

+1.000
TECH TEAMS

+6.500
REPOSITORIES

+1,6 M
DEPLOY / YEAR

+6.500 M
VISIT WEB / YEAR

RESPONSABILITY MODEL

DEV

PLATFORM TEAM

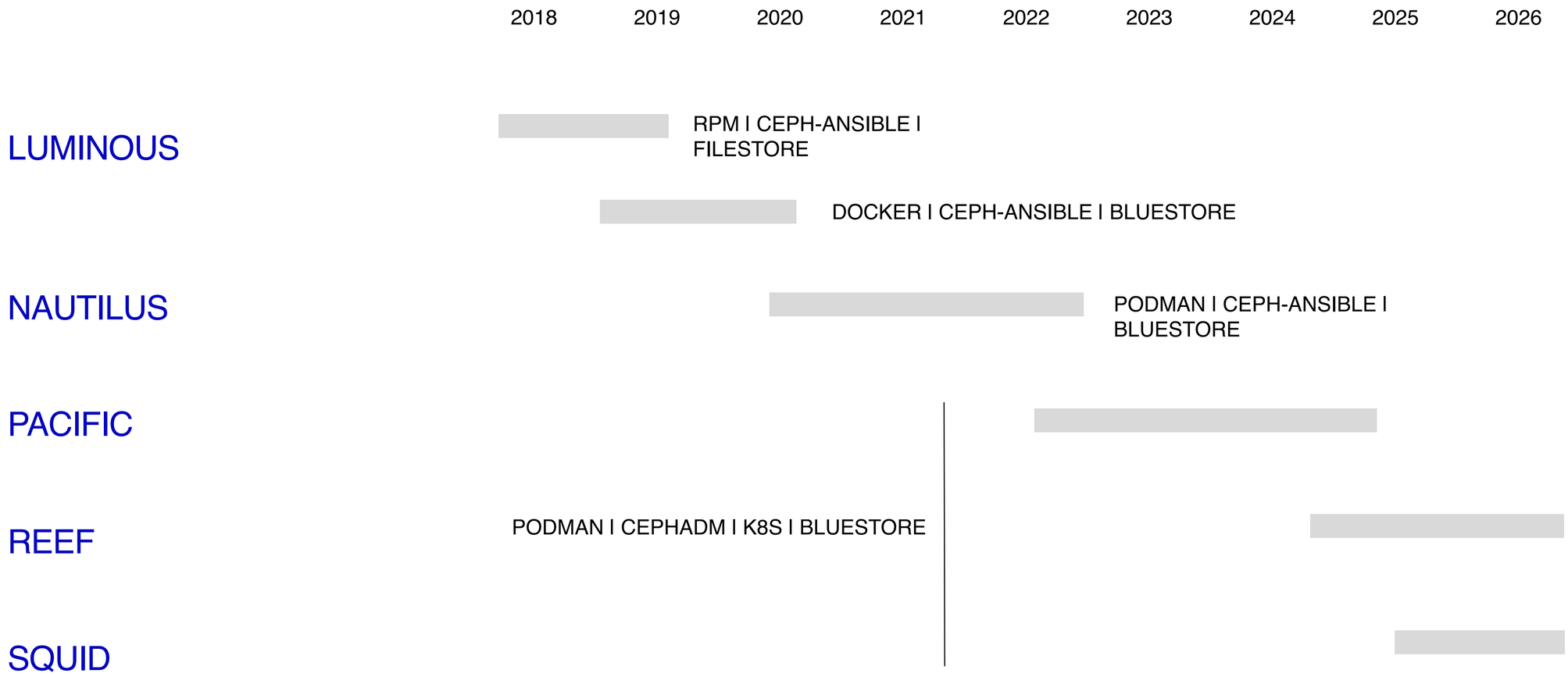
CODE

FRAMEWORK
CI/CD
MONITORING
INFRASTRUCTURE

APP.YAML

NAME: MI-APP
PLATFORM: PRODUCTION
REPLICAS: 3
CPU: 4
MEM: 2
PORTS: 8080

DEPLOYMENT
SERVICE
INGRESS/GATEWAY
PVC
HPA
...


2018
2019
2020
2021
2022
2023
2024
2025
2026
LUMINOUS
RPM I CEPH-ANSIBLE I FILESTORE
DOCKER I CEPH-ANSIBLE I BLUESTORE
NAUTILUS
PODMAN I CEPH-ANSIBLE I BLUESTORE
PACIFIC
REEF
PODMAN I CEPHADM I K8S I BLUESTORE
SQUID

ARQUITECTURE

02

SEWING AI

FROM IDEA TO PRODUCTION-READY WEB APPS -
WITH LESS FRICTION ACROSS PRODUCT, UX, AND DEVELOPMENT.


SEWING AI
WHAT DO YOU WANT TO CREATE?
DESIGN
DEVELOPMENT
Ask Sewing AI to build...

REQUIREMENTS

KUBERNETES READY

Must run natively on Kubernetes using operators, CRDs, and standard pod lifecycle

SCALABLE

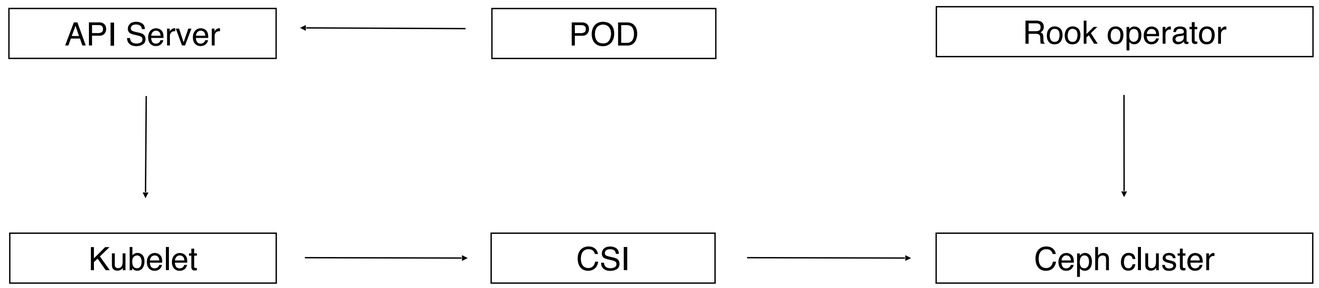
Must handle 10 to 200+ concurrent developer workspaces without performance degradation

PORTABLE

Must deploy on any Kubernetes distribution

AGNOSTIC INFRAESTRUCTURE

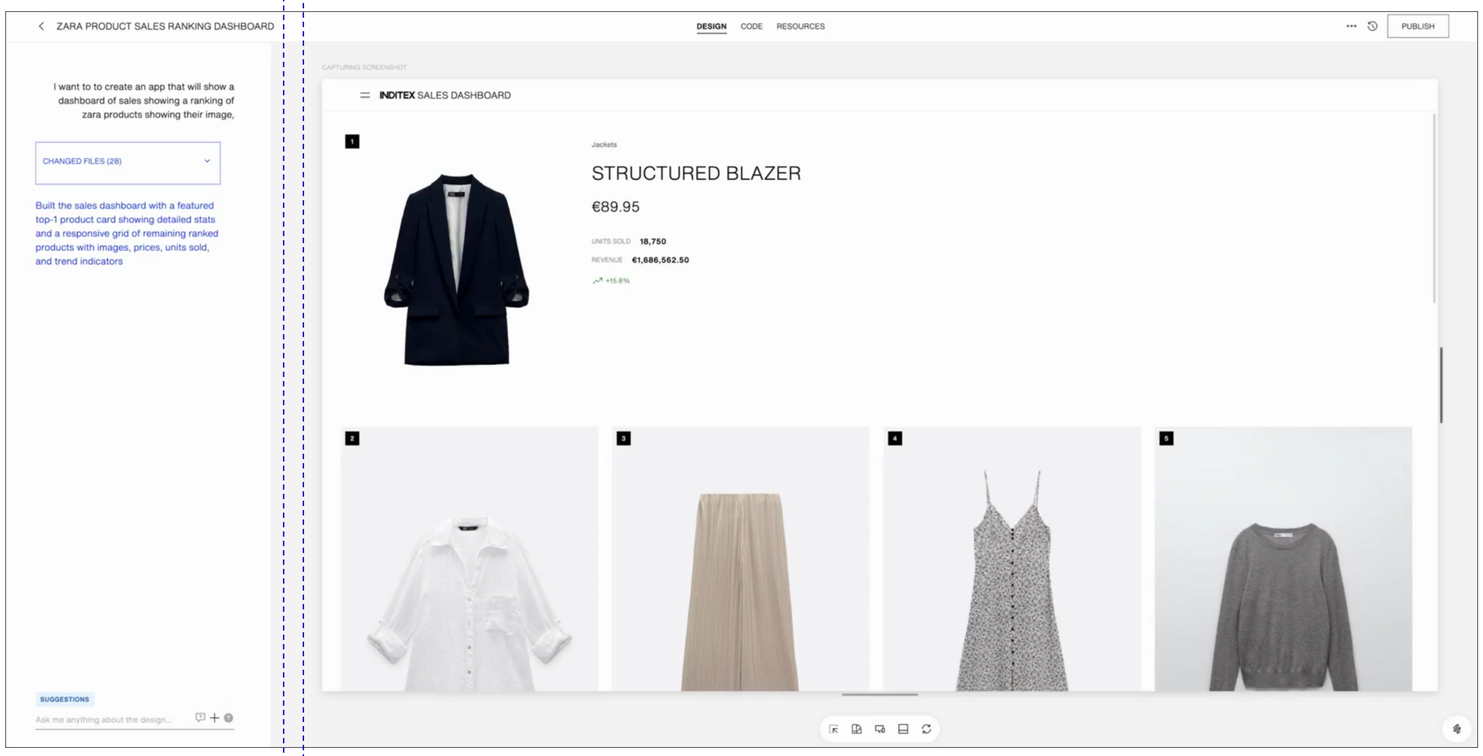
No lock-in to a specific cloud provider



CLOUD+K8S+ROOK

ROOK IS A KUBERNETES OPERATOR THAT AUTOMATES THE DEPLOYMENT, CONFIGURATION, AND MANAGEMENT OF CEPH STORAGE CLUSTERS

Agent



K8S

Dev workspace

Ceph

</>
</>
</>
</>

HOW DOES A WORKSPACE WORK?

01

START WITH A BASIC TEMPLATE

CLONE A PRE-DEFINED PROJECT SKELETON

02

SETUP ENVIRONMENT

CONFIGURE RUNTIME, ENV VARS, ETC

03

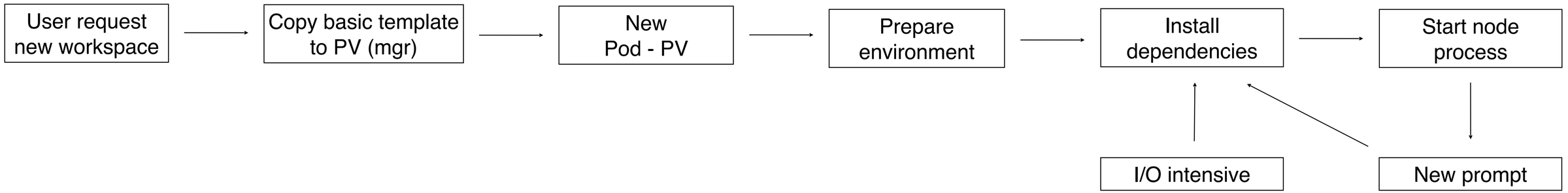
INSTALL DEPENDENCIES

(P)?NPM INSTALL

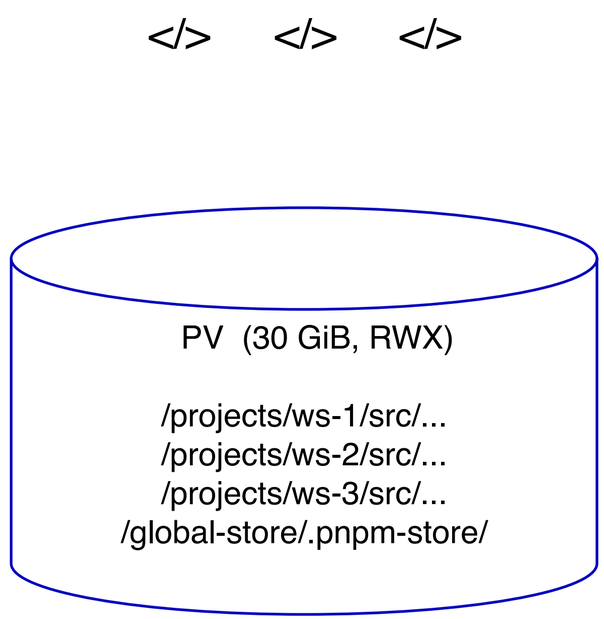
04

WAKE UP NODE + HMR

START THE DEV SERVER WITH HOT MODULE REPLACEMENT



FIRST APPROACH FOR A NEW WORKSPACE


</>
</>
</>
PV (30 GiB, RWX)
/projects/ws-1/src/...
/projects/ws-2/src/...
/projects/ws-3/src/...
/global-store/.pnpm-store/

PV CONTENT

BOTTLENECK

03

FROM ~50 SEC TO TIMEOUT

SELINUX RELABELING

PROCESS OF ELIMINATION

FINDING THE BOTTLENECK

37 M REQ IN 15’

THE NODE_MODULES PATH RESOLUTION TAX

CACHE EVICTION + CAPS RECALL

SELF-DDOS

FROM ~50 SEC TO TIMEOUT

STARTING POINT

Rook by default:

- 3 OSD
- 1 mds active
- 2 mgr
- 3 mons

VALIDATING CEPHFS FOR DEVELOPER WORKSPACE

Load testing before GO

TEST METHOD: INCREMENTAL

Launch workspaces progressively: 1 … 200 concurrent

SELINUX RELABELING

FIRST WORKSPACES WORK FINE, NOT THE SECOND ONE

When a pod mounts a volume, kubelet applies SELinux labels recursively to every file

relabeling = stat + setxattr on every single file

PROCESS OF ELIMINATION

3 WORKSPACES WORKS FINE, NOT 20

- Load test at 20 concurrent workspaces – workspaces do not start at time – kill by kubelet
- Remember: only 3 OSDs and 1 MDS at this point
- Systematic check:
  - NETWORK: NETWORKING, LATENCY < 1MS
  - OSD THROUGHPUT: 3 SSDS STILL HAD IOPS
  - KUBERNETES SCHEDULING: POD PLACEMENT < 2S
  - PVC provisioning: CephFS subvolume creation < 500ms
- Ceph health detail: MDS_SLOW_REQUESTS
- MDS daemon: single CPU core at 100%

FINDING THE BOTTLENECK

PERF DUMP

ceph daemon mds.<name> perf dump

Hard to see in terminal, we built a dashboard to draw n_requests

- MDS is single-threaded for request processing
- With 3 OSDs backing metadata pool: ~2ms per RADOS read
- Saturation point for 1 MDS rank: ~8,000 ops/s
- The default deployment is not sized for metadata-heavy workloads

37 MILLIONS REQ IN 15 MIN

200 WORKSPACE SCENARY

- Worst-case test: 200 workspaces launched in 10-min window
- In first 180s (pnpm install) - peak of traffic:
  - Link: ~42,000 ops
  - lookup: ~30,000 ops
  - CREATE: ~9,000 OPS
  - mkdir: ~7,000 ops
  - GETTADR: ~5,000 OPS
  - open: ~1,000 ops
- Aggregate: ~37M req ops in 15 minutes

THE NODE_MODULES PATH RESOLUTION TAX

NPM VS PNPM

The problem with npm:
- npm install creates a full copy of every dependency per workspace
- Typical project: ~100,000 files in node_modules folder

pnpm with content-addressable store on a shared volume:
- pnpm uses a global content-addressable store + hard links
- All workspaces on the same PV share the same physical files

Hard links = same inode, no extra metadata - MDS sees one entry, not N copies

Massive reduction in:
- Create ops (files already exist in store)
- Stat ops (no permissions to set)

MDS cache pressure (fewer unique inodes to track)

First `pnpm install` populates the store; subsequent installs are mostly hard link operations

Requires all workspaces on the same volume

CACHE EVICTION + CAPS RECALL

LATENCY

• Cache eviction: When mds_cache_memory_limit is too small for the working set, the MDS constantly evicts inodes

• Caps recall: Shared dependencies (pnpm) are accessed by all workspaces simultaneously, MDS must revoke capabilities from client A to grant them to client B, serialized overhead.

• The compounding effect: Evicted inode needs re-fetch → re-fetch requires caps → caps must be recalled from another client → double the latency on an already saturated daemon

• MDS_CLIENT_RECALL

SELF-DDOS

DDOS

MDS saturated (queue > 4,000)
→ client ops timeout (session_timeout: 60s)
→ CephFS mount hangs inside container
→ readiness probe fails → kubelet restarts pod
→ fresh pod = cold cache = npm install from scratch
→ MORE metadata ops → queue grows → cascade

The system made itself worse
Pod restart = CephFS session teardown + re-open + all caps re-requested

SCALE UP

04

LAYER 0

AVOID SELINUX RELABEL

New pods must be created with this option:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    io.kubernetes.cri-o.TrySkipVolumeSELinuxLabel: "true"
spec:
  securityContext:
    fsGroupChangePolicy: OnRootMismatch
```

LAYER 1

INCREASE CACHE MEMORY

- First attempt: tune without adding hardware — see how far we can go
- Changes applied:

```
# Double the cache to hold more inodes
ceph config set mds mds_cache_memory_limit 8589934592   # 8 GiB (was 4 GiB)<
```

LAYER 2

CRD

The big leap: from 1 active MDS to 16 active + 16 standby-replay

```
# CephFilesystem CRD:
  metadataServer:
    activeCount: 16
    activeStandby: true
    resources:
      requests:
        cpu: "4"
        memory: 16Gi
      limits:
        cpu: "4"
        memory: 16Gi
```

MDS cache per rank: 8 GiB - Total: 128 GiB (vs 4 GiB before)

LAYER 3

INCREASE OSD

OSD layer needed to support multi-MDS metadata pool IOPS
Scaled storage infrastructure:

```
# StorageCluster CRD
storageDeviceSets:
  - name: deviceset-managed-premium-v2-csi
    count: 9            # was 1
    replica: 3          # 9 × 3 = 27 OSDs
    dataPVCTemplate:
      spec:
        resources:
          requests:
            storage: 128Gi
        storageClassName: managed-premium-v2-csi
```

- 9 storage nodes × 48 vCPU / 192 GiB RAM each
- 27 OSDs across 3 Azure availability zones (1/2/3)
- OSD resources: 3 CPU / 16 GiB per OSD
- Metadata pool: 3× replication, failure domain = zone, 32 PGs
- OSD commit latency: 0-2ms (Azure Premium SSD v2)
- Result: metadata pool IOPS capacity went from ~15k to **~135k**

LAYER 4

PIN

- The key: subvolume export pinning, NOT the automatic balancer
- Architecture:

16 CephFS subvolumes (PVCs, 30 GiB each, RWX)
- ├ subvolume-0 → export_pin = MDS rank 0
- ├ subvolume-1 → export_pin = MDS rank 1
- ├ subvolume-2 → export_pin = MDS rank 2
- ├ ...
- ├ subvolume-15 → export_pin = MDS rank 15

200 workspaces distributed across 16 subvolumes
- → ~12-13 workspaces per subvolume
- → ~12-13 workspaces per MDS rank (deterministic)

- Each workspace pod mounts one of the 16 PVCs (assigned by slot/round-robin)
- Export pin guarantees: **this subtree always goes to this rank — no balancer needed**
- Result: deterministic, predictable distribution — no balancer lag, no rebalance storms

ceph fs subvolume pin storagecluster-cephfilesystem csi-vol-56df447c-149c-4091-82f1-f5d524893906 export 0 --group_name csi

FINAL STAGE

FINAL TEST

Current production cluster state:

- HEALTH_OK, 216 CephFS clients connected
- 32.49M (10.91M metadata, 21.58M data)
- 47.2 M inodes cached across 32 MDS daemons (16 active + 16 standby-replay)
- Rados latency: 1.67ms avg
- Slow replies: 0
- Peak Caps: 32k

SUMMARY

01

SELINUX

ONROOTMISMATCH

02

MDS TUNING

INCREASE MEMORY ONLY

03

SCALE OSDS

FROM 3 TO 27 OSDS

04

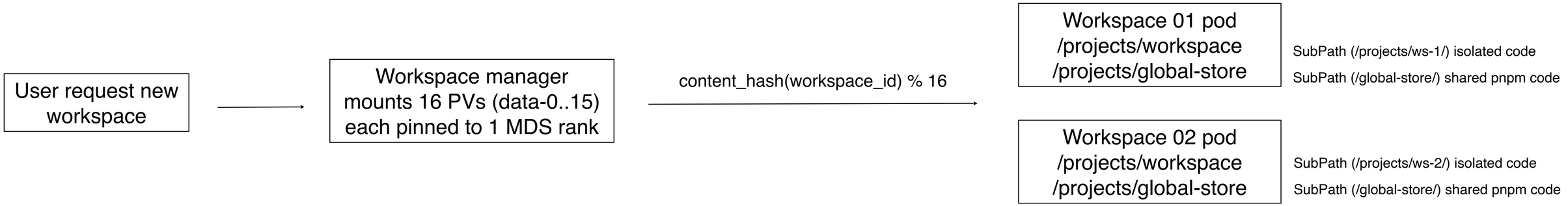
SCALE MDS

CREATE 16 ACTIVE MDS

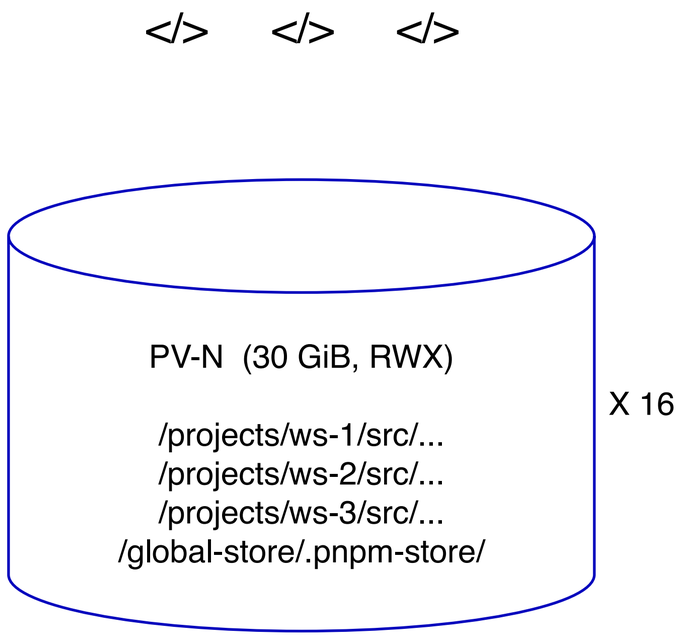
05

SUBVOLUME PINNING

DETERMINISTIC SHARDING



FINAL APPROACH, ISOLATED



PV CONTENT FINAL APPROACH

OBSERVABILITY

05

TRACK OVER TIME

PROMETHEUS RULES

- External Prometheus with 1-year retention
- Detect patterns - Cache hit ratio drift, inode growth rate, seasonal bursts (Monday mornings, team onboarding, internal training, etc)
- Proactive scaling trend shows MDS saturation approaching in 6 weeks?
- Correlate infra metrics with business events like new features big repo = inode explosion. The timeline tells the story

OPEN CASES & LESSONS

06

OPEN CASES

TO-DO

- RAM cost: 280 GiB just for MDS
  - 32 pods × 8.8 GiB RSS average
  - Is 16 ranks optimal, or could 8 ranks with larger cache achieve similar results?
  - Trade-off: more ranks = better parallelism, but more memory and operational complexity
- Manual pinning = manual management
  - Adding/removing subvolumes requires re-pinning
  - If workspace count grows beyond 200, do we add more subvolumes + ranks?
  - No auto-scaling today
- Uneven load distribution
  - Not all subvolumes carry equal load - some teams are more active than others
  - Periodic re-balancing of workspace-to-subvolume assignment may be needed
- The Node.js ecosystem keeps growing
- Node Ready ≠ ready to mount storage – daemon set missing at the first seconds
- Storage cannot decrease when the environment does not have load

MAIN LESSONS

WHAT WE HAVE LEARNED

- Test at target scale BEFORE go-live
- ROOK defaults ≠ production-ready
- Do not trust the balancer for bursty workloads
- Scale the full stack, not just the hot spot
- The cache miss penalty increase latency
- Observability, Observability and Observability

Q&A

07